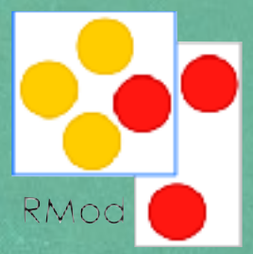
Modular Stateful Traits in Pharo

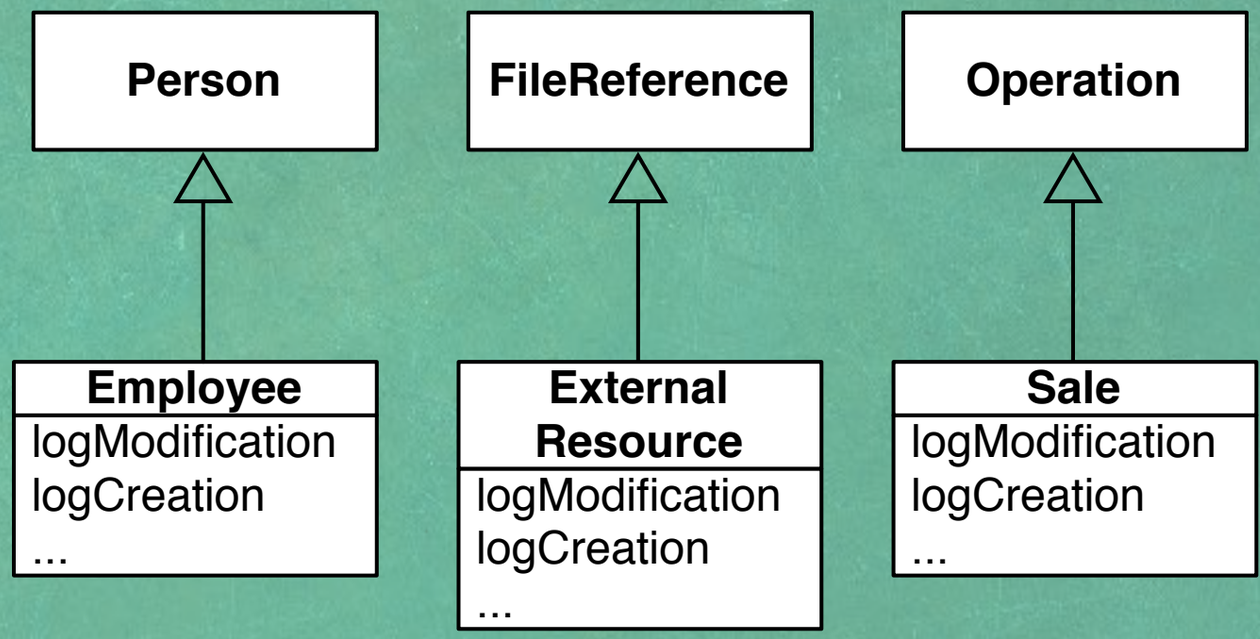
Pablo Tesone

IMT - Lille-Douai / RMod INRIA



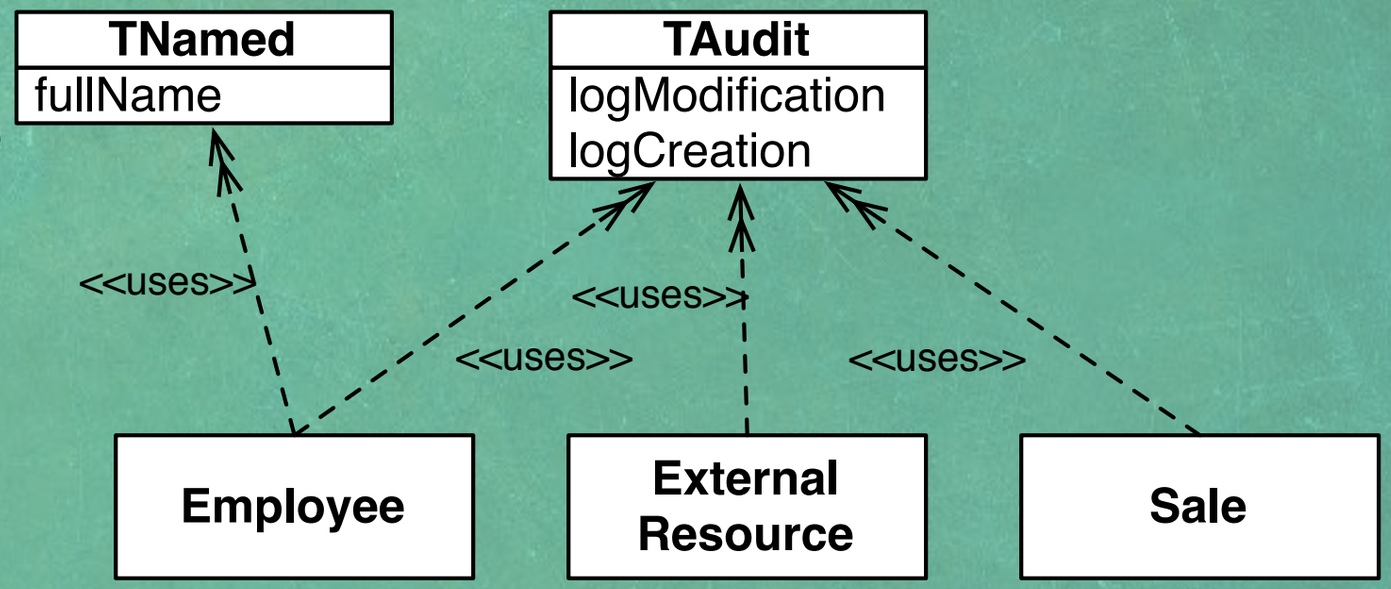
Traits

- We have duplicated behaviour.
- Present in different hierarchies.
- Inheritance is not enough



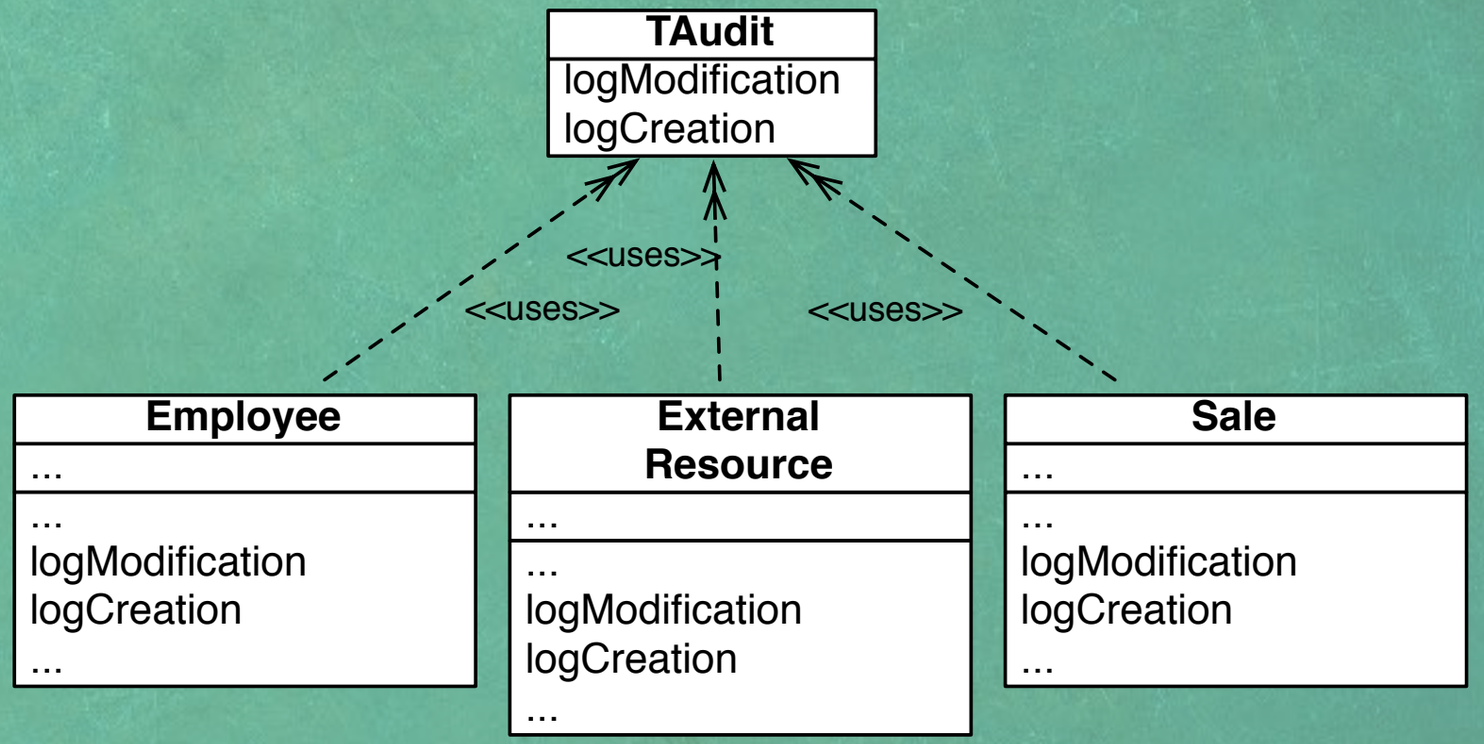
Traits (2)

- Reuse of behaviour
- Share behaviour outside the hierarchy of classes.
- Reduce the duplication of code.
- Manual resolution of conflicts.
- Not affecting the subclassing.



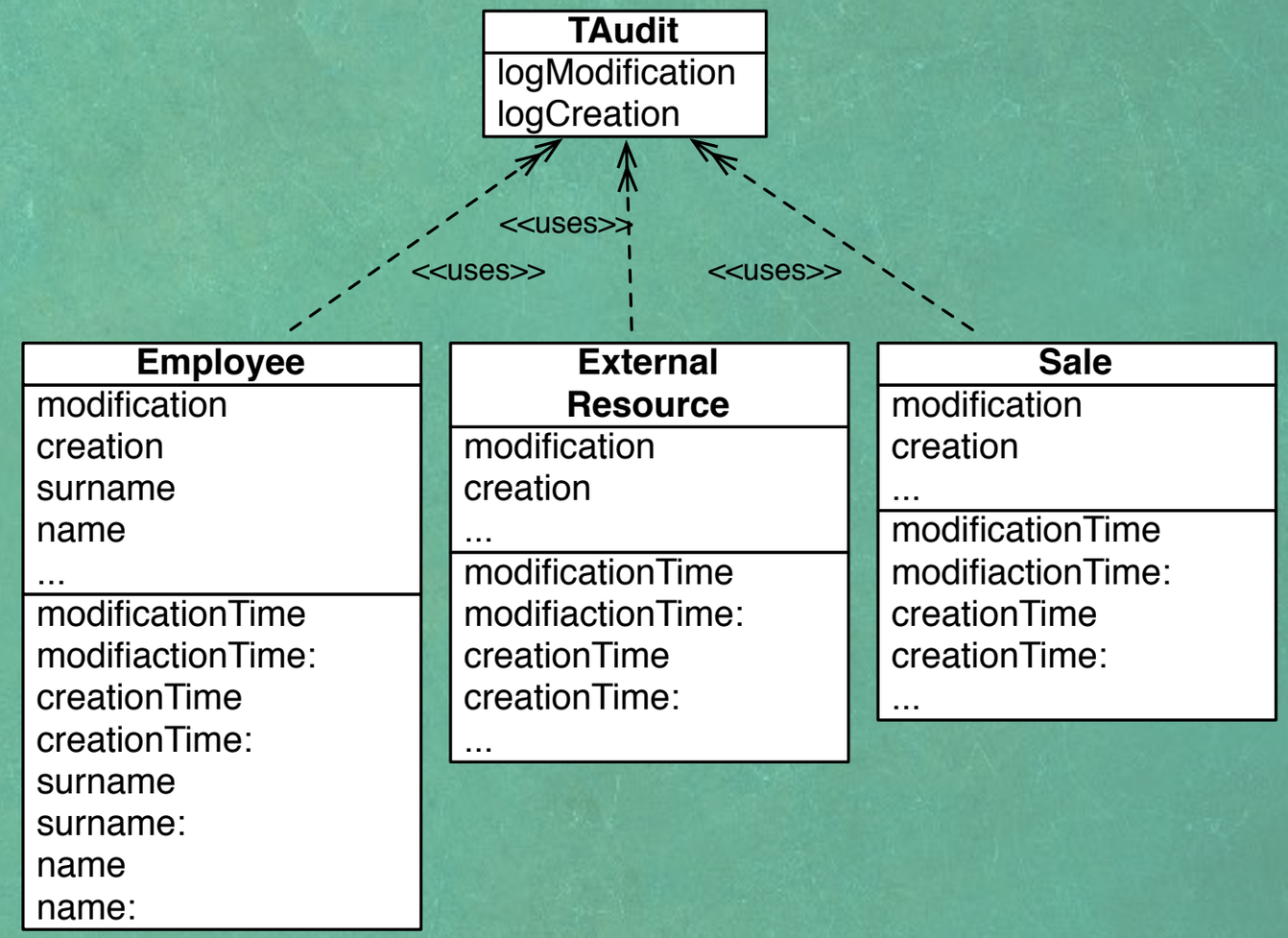
Some Implementation Details

- Traits are flattened.
- Classes have copy of the methods
- Handled by the traits implementation… not by the programmer.
- Transparent to the Runtime



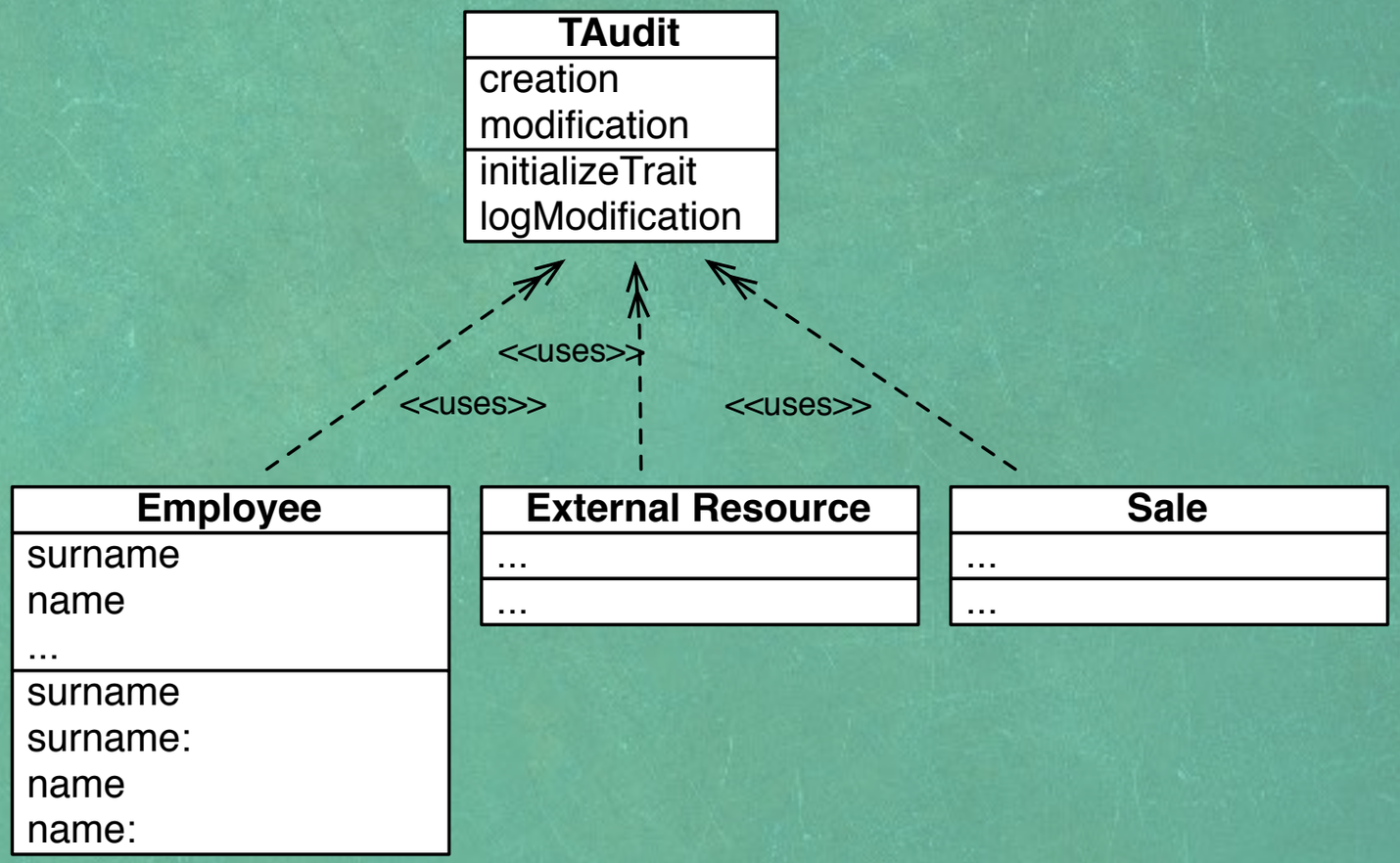
But not perfect…

- Only can interact with the instance through messages.
- Needs of glue code (usually accessors).
- Cannot define instance variables (slots)



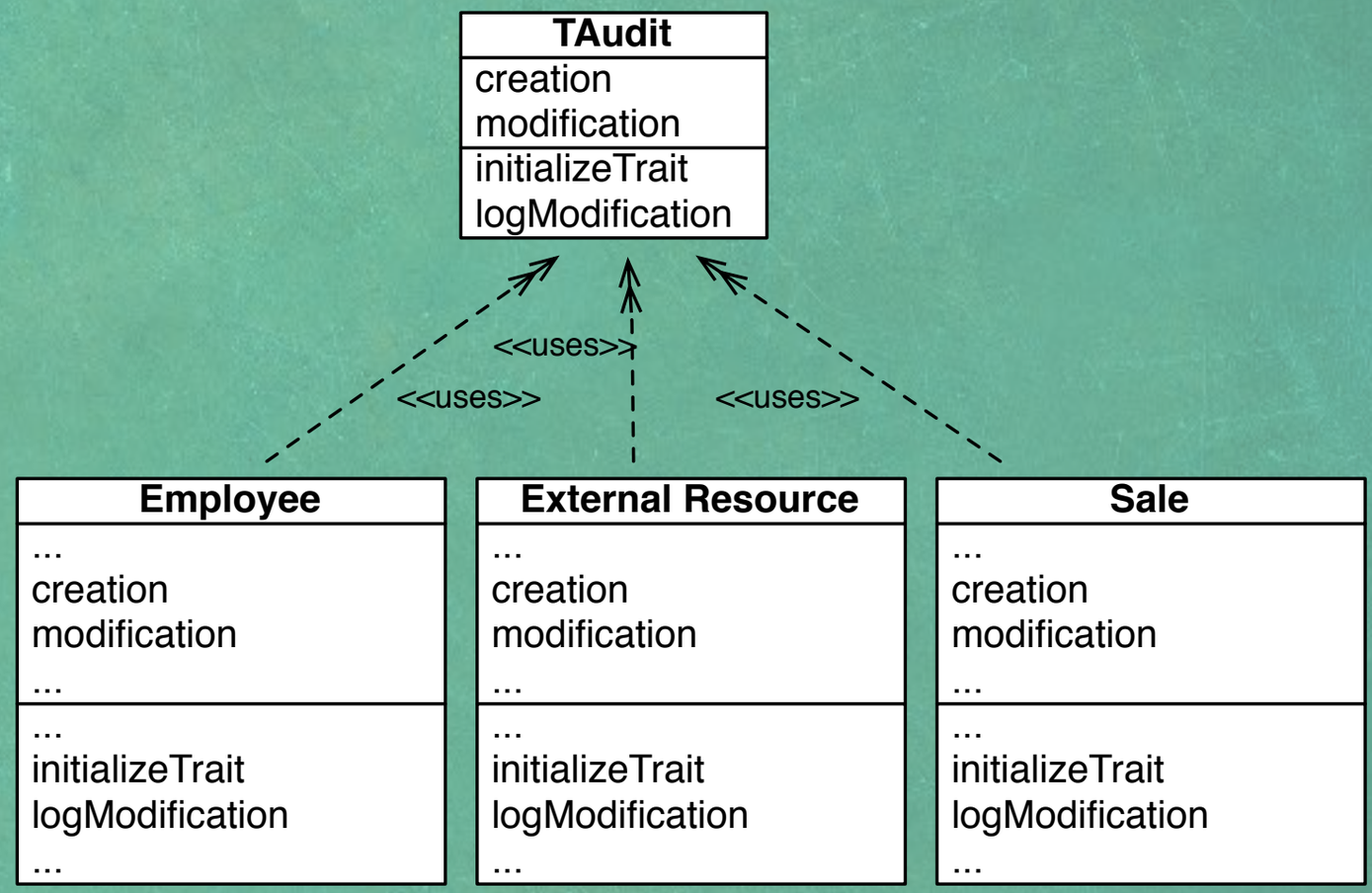
Stateful Traits

- Reuse of slots.
- Improving the Initialisation
- We reduce the glue code.
- Adding new operations to handle conflicts.



Stateful Traits (2)

- Flattened Methods.
- Flattened Slots.
- Everything is compiled correctly.
- Methods defined in a trait accesses slots defined in it directly.

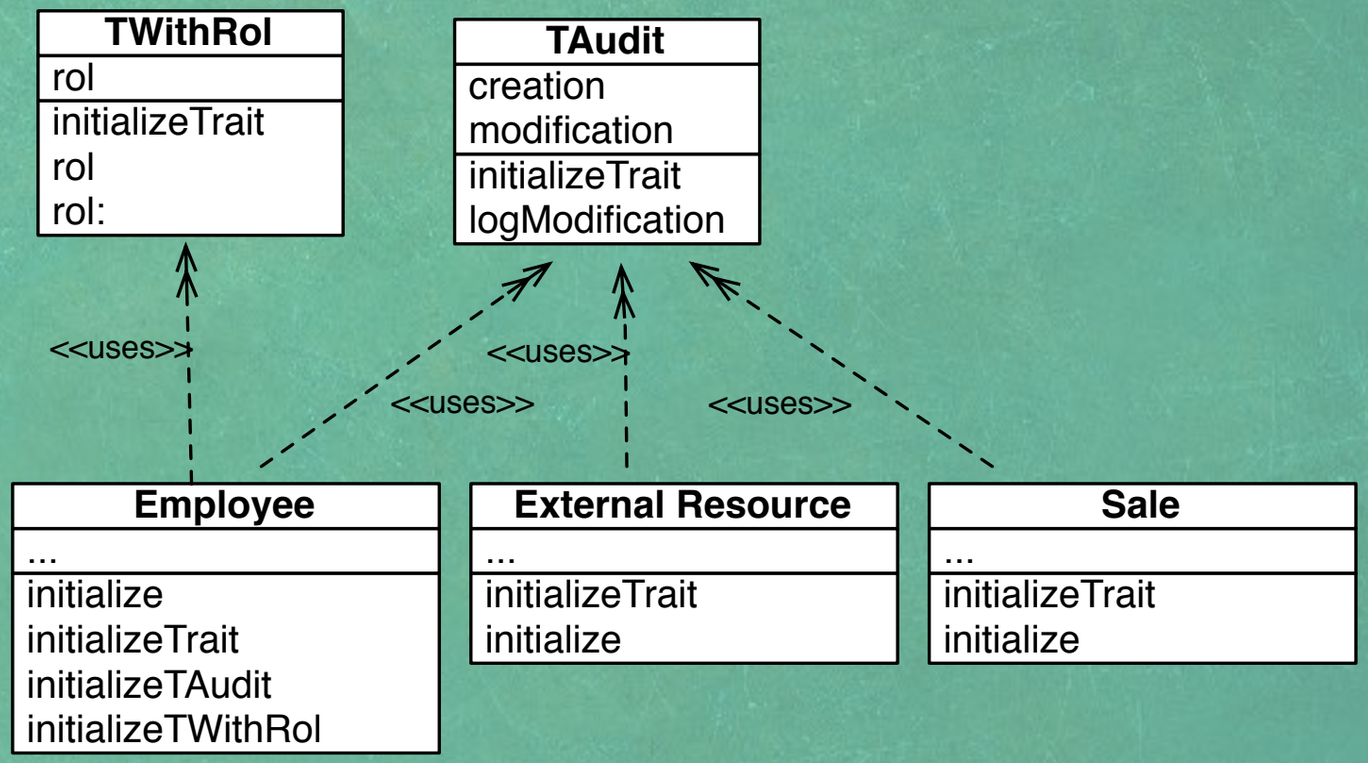


Initialisation of instances

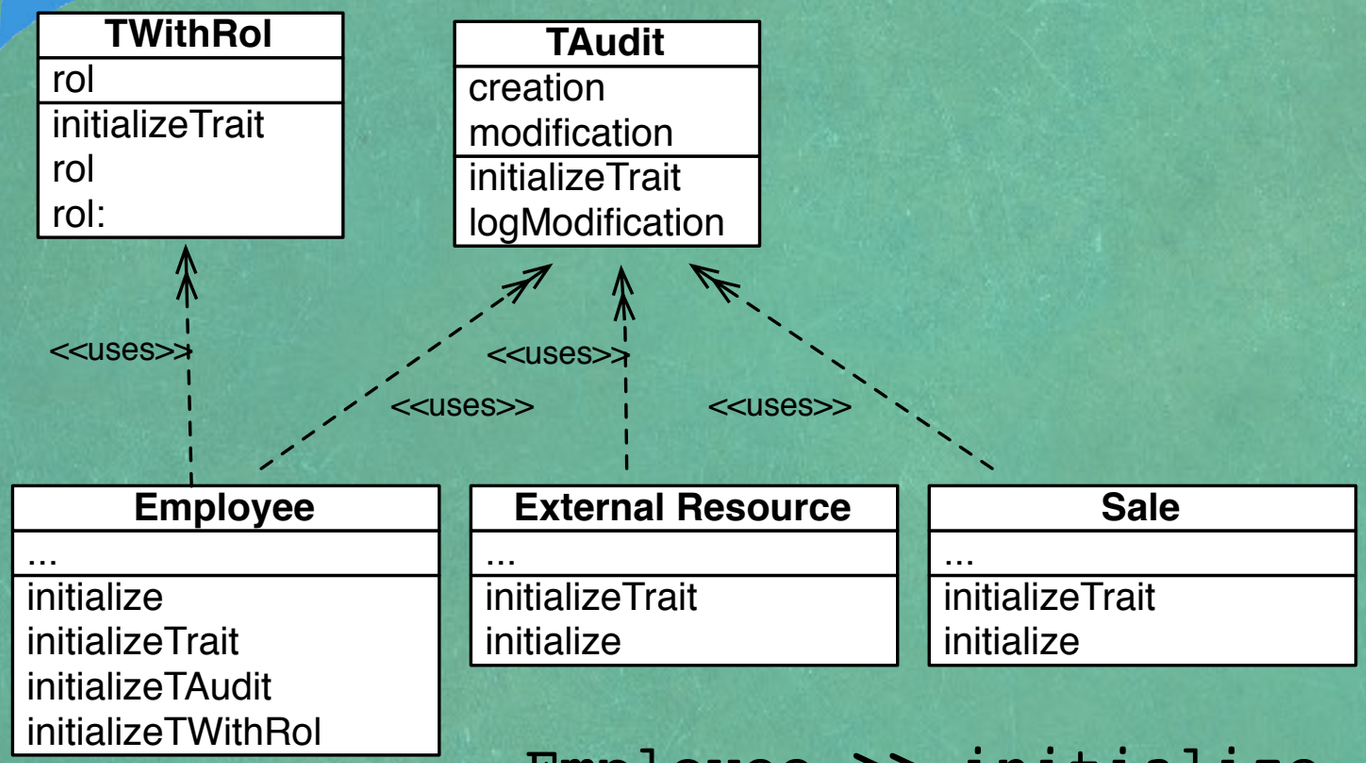
- Traited classes have another initialisation method.

- Perform the initialisation of all the traits.

- Generated if missed by the traits mechanism.



Initialisation of instances



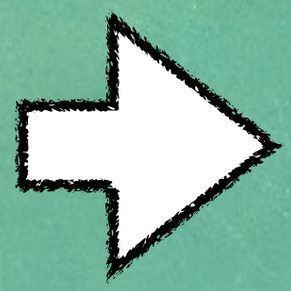

```
Sale >> initialize

  super initialize.
  self initializeTrait
```

```
Employee >> initialize

  super initialize.
  self initializeTrait
```

```
Employee >> initializeTrait

  self initializeTWithRol.
  self initializeTAudit
```

New Trait Composition Operations

Adding Slots brings new conflicts.



- Alias Slot
- Remove Slot
- Merge Slots


That's not new!!! Kill him!!

Also Fix other problems

Pharo 6 Traits

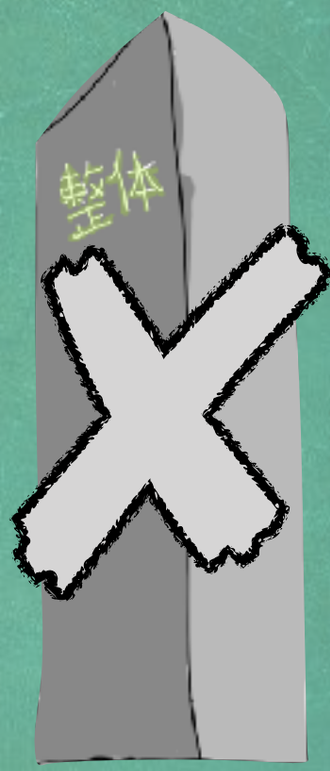
Monolithic Impl.

Not Extensible

Limited Polymorphic w/classes

Modular Implementation

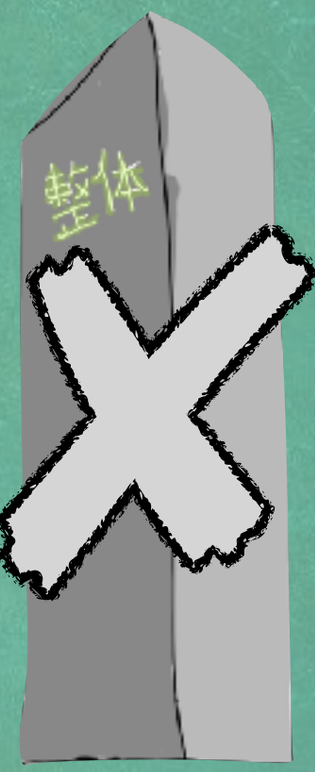
- The kernel does not know nothing about traits.
- They are loaded after.
- Reducing Codintional Code.



Modular Implementation

Pharo 6

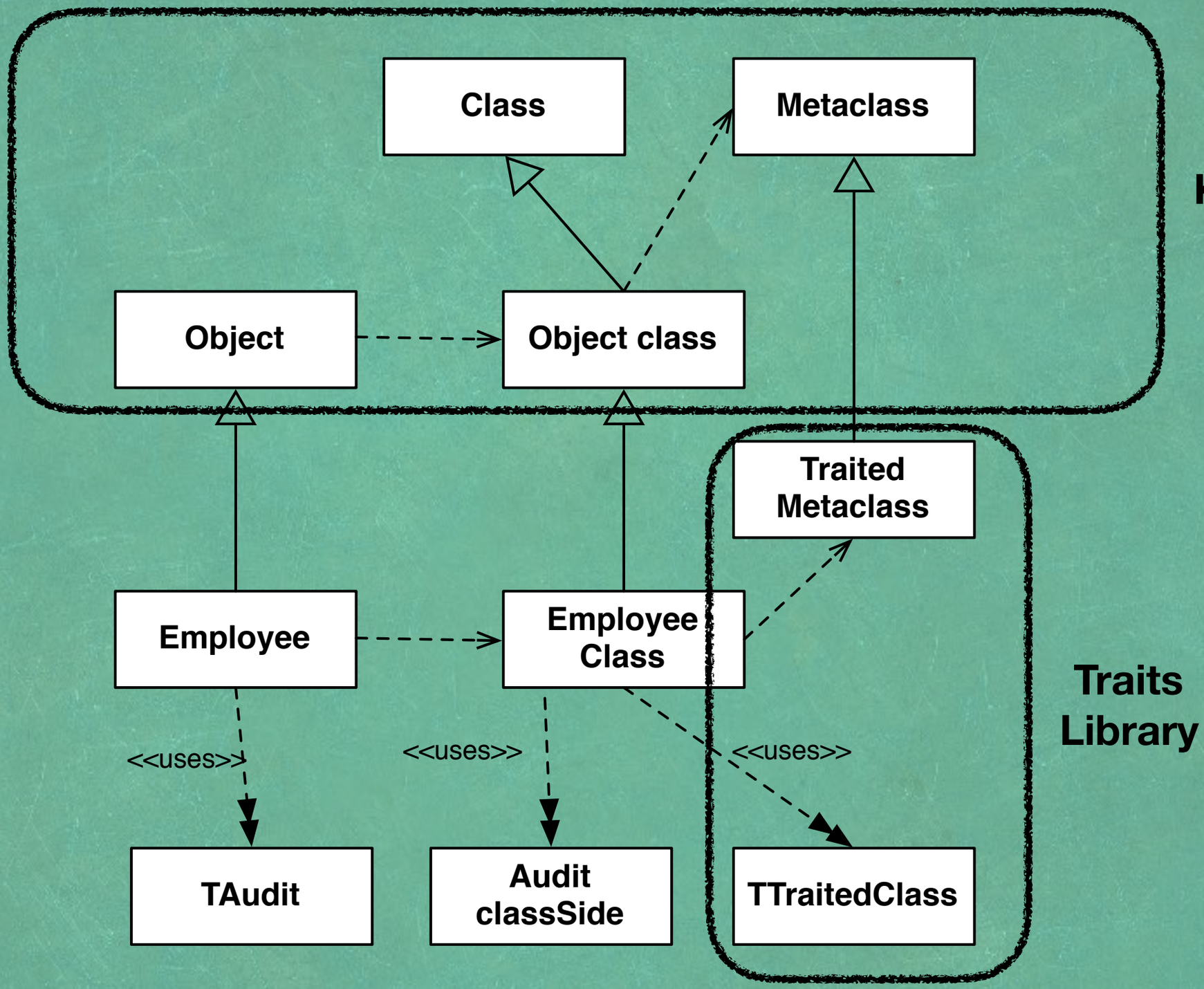
```
Behavior >> includesBehavior: aClass

    self isTrait ifTrue: [ ^false ].
    ^self == aClass or:[self inheritsFrom: aClass]
```



Pharo 7

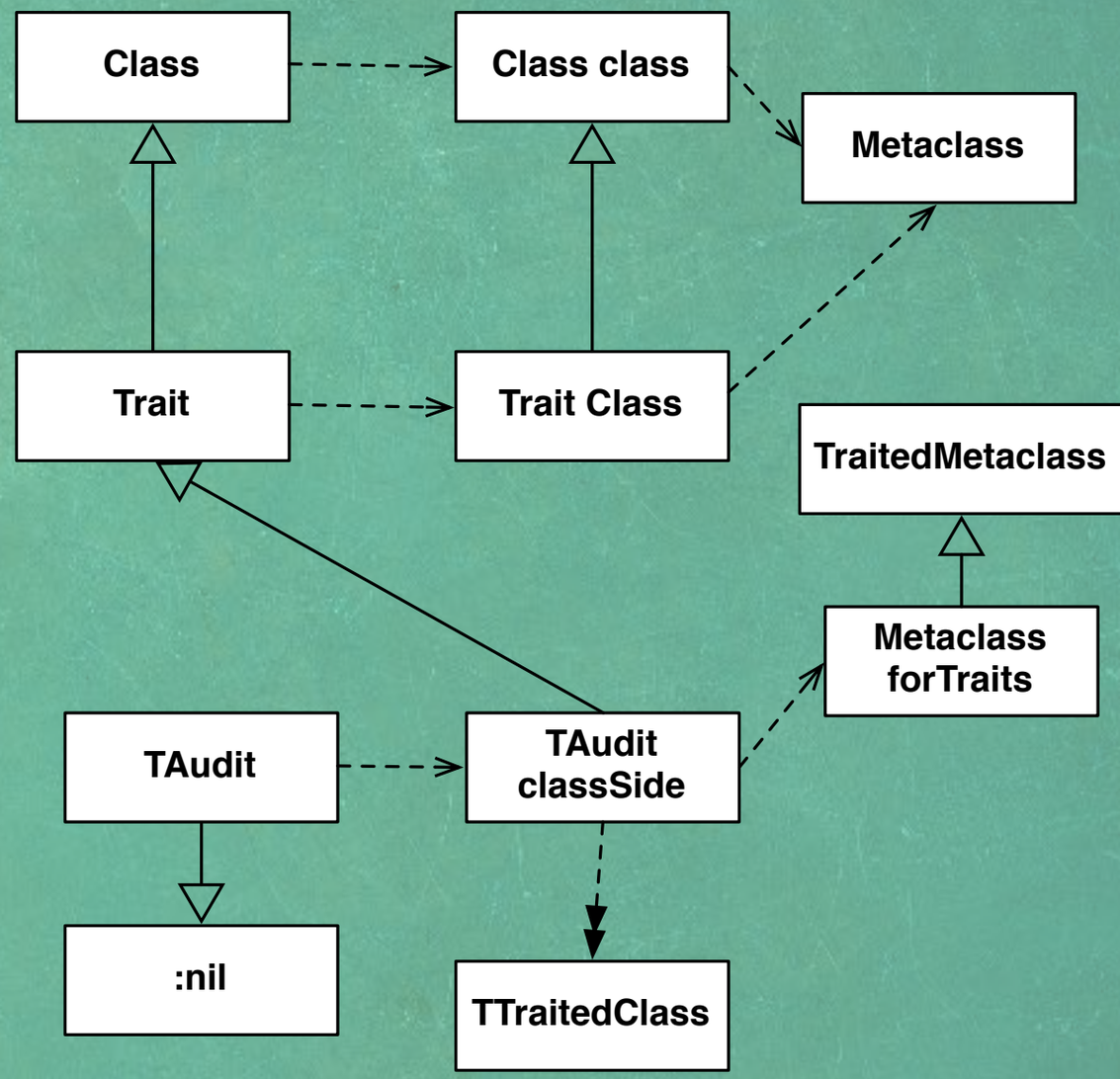
```
Behavior >> includesBehavior: aClass

    ^self == aClass or:[self inheritsFrom: aClass]

Trait >> includesBehavior: aClass

    ^false
```

Solved by Polymorphism



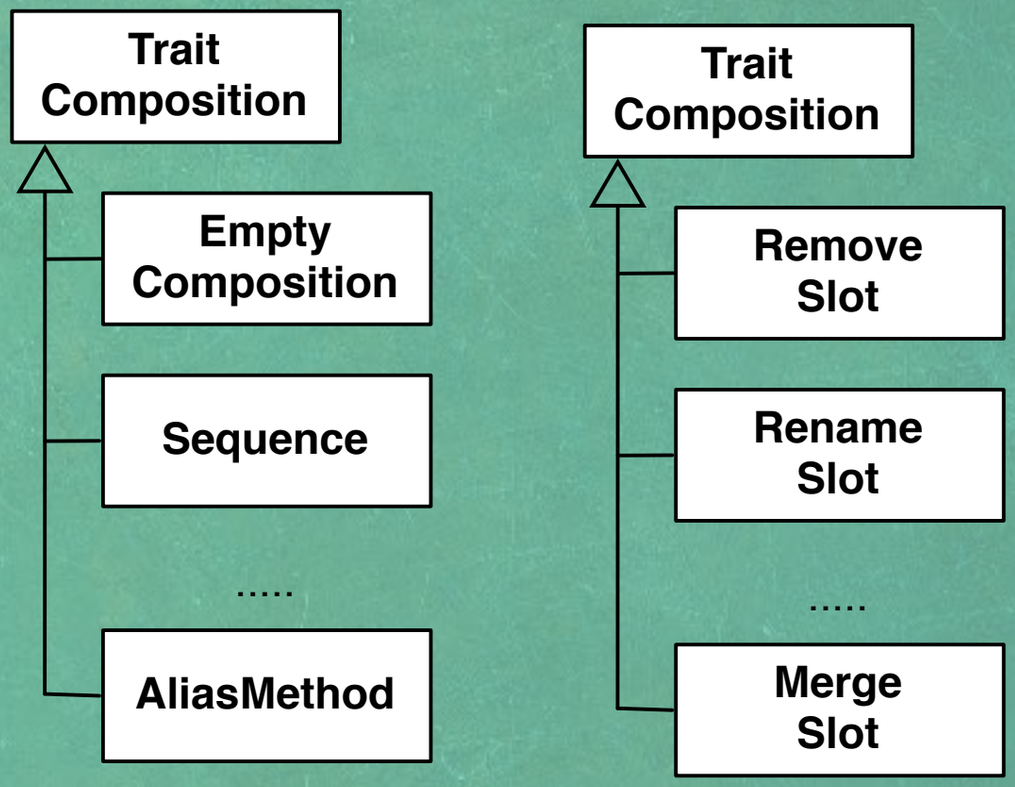
Polymorphism w/Classes

- The traits are traited classes…..
- But you cannot instantiate them.
- We only provide the different behaviour.
- Simplifies the tools



Extensible Algebra

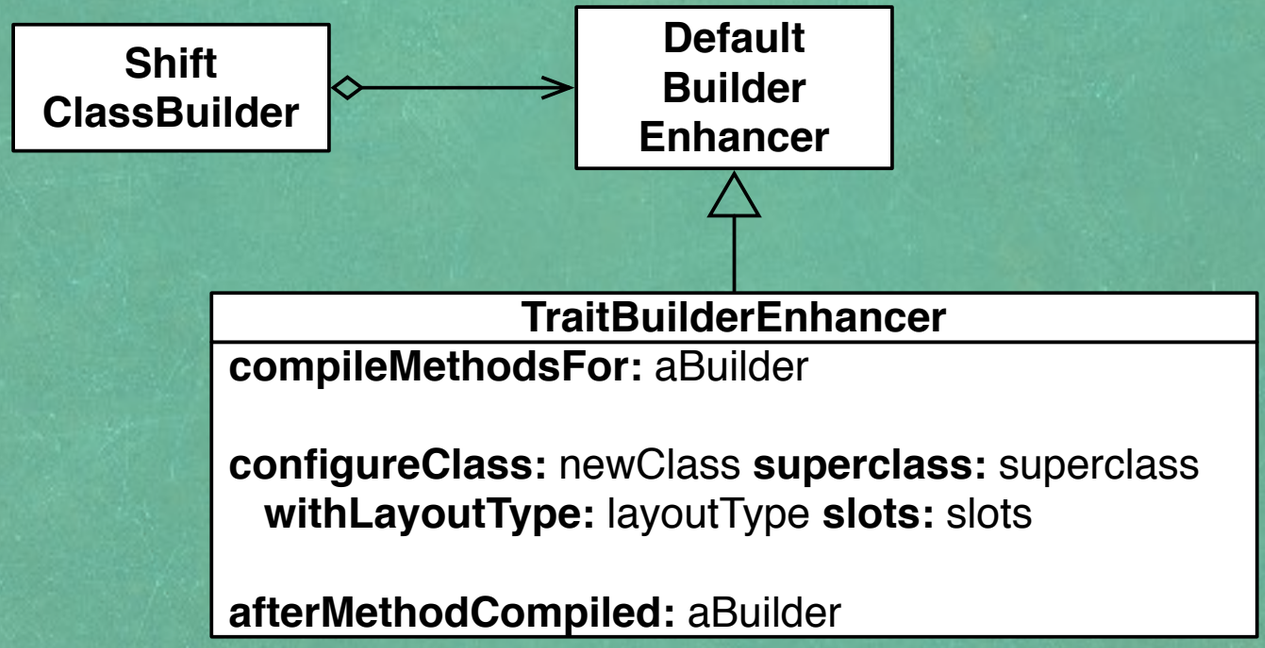
- We need the algebra to solve conflicts.
- New operations to handle slots:
  - Rename
  - Remove
  - Alias
- Also the algebra easily extensible.



Extensible Class Builder

- The class building process has been modified. Allowing external extensions.

- All the creation of classes with traits are extension methods.



Does it scale?

- The methods and slots are flattened
- The slots are compiled in the target class.
- The initialisation code is chained.
- During runtime, there is no difference.
- 100 % Backward compatible.
- No VM Changes or needed support.

Bonus: simplification of MetaObject - Protocol

- Same MOP with classes.
- Classes, Traits (whatever) are only source of slots and methods.
- Better API for selectors, localSelectors and slots and localSlots.
- Simplifying the tools.
- System Traits deprecated (TClass, TBehavior, …)
- Still in progress…. so many tools to update.

For the same price…. (?)

Kernel Reduction

Lines	22,606	15 %
Methods	2,897	21 %
Bootstrap Process	5 min	30 %
Overall Pharo Build	4 min	20 %

Lines	6,557
Affected Packages	89

Tool Simplification

Conclusion

- Stateful Traits!
- Modular Solution (faster bootstrap, you can avoid it).
- Extensible

THANKS!!!

- Simpler MOP / Polymorphism with classes.
- Same performance!
- 100% Backward Compatible